# Computing Schrödinger's Equations

Due Date: Week 9 Monday, 20 March, 5 pm

| | | | |
|---|---|---|---|
| Name: | ______________________________ | Principal Quantum Number: | _______ |
| | ______________________________ | | |
| | ______________________________ | | |

## Objectives

a. Understand the application of coordinate conversion in the Schrödinger's equation.
b. Apply both the Legendre and Laguerre polynomials to solve the angular and radial forms of the Schrödinger's equation respectively.
c. Calculate the complete wavefunction solution from the radial and angular solution.
d. Plot and visualize the probability density functions and the shapes of these orbitals.

## Instructions and Weightage

For this assignment, you are required to work in assigned groups of 2 or 3. This assignment is split into two sections:
Part A: Theory (Chemistry) Segments
Part B: Coding (Digital World) Segments

In this assignment, choose the principal quantum number $n$ (2, 3 or 4) that your team will be working on. You will need to complete this assignment for all $l$ and $m$ (or $m_l$ in your Chemistry notes) values for that chosen $n$ i.e. for $n = 2$, solve for $l = 1$ and $m = -1, 0, 1$ and for $n = 3$, solve for $l = 1, 2$ and $m = -1, 0, 1$, $m = -2, -1, 0, 1, 2$ and so on.

This assignment is worth 30 points or 5.5% of the final Physical Chemistry grade and 2% of the final Digital World grade. A bonus of up to 15 or 30 additional points is available:

- No additional points if you attempt $n = 2$.
- 15 additional points if your attempt in solving for all orbitals in $n = 3$ is completely correct.
- 30 additional points if your attempt in solving for all orbitals in $n = 4$ is completely correct.

## Assignment Submission

You are required to notify your Chemistry instructors of your choice of principal quantum number $n$ by the end of Week 6 Friday, 3 March, 5 pm.

You are required to submit the following to your Chemistry instructors on Week 9 Monday, 20 March, 5 pm for all $m$ numbers in $l = 1$ (compulsory) for your chosen principal quantum number $n$:

(i) Answers to the Chemistry Section (Part A) – write your answers on a separate piece of paper and attach to this handout if the space given in this handout is insufficient
(ii) 3D Plots (Part B)
(iii) If your team chooses $n = 3$, you are required to submit the above items (i) and (ii) for all $m$ numbers in $l = 1, 2$ in order to be considered for the bonus points
(iv) If your team chooses $n = 4$, you are required to submit the above items (i) and (ii) for all $m$ numbers in $l = 1, 2, 3$ in order to be considered for the bonus points.

## References

**It is highly recommended that you read References (b) to (g) as it may prove to be very useful for this assignment.**

a. Introduction to Quantum Mechanics (Second Edition), David J. Griffths
b. From the time independent Schrödinger's equation in 3D to spherical harmonics, https://www.youtube.com/watch?v=bqYMUDxD3WQ
c. Time independent Schrödinger's equation in 3D radial behavior, https://www.youtube.com/watch?v=Nq72tUEtbfM
d. Matplotlib 2D/3D plotting tutorials, http://matplotlib.org/users/pyplot_tutorial.html, http://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html
e. Python scripting for 3D plotting, http://docs.enthought.com/mayavi/mayavi/mlab.html
f. Hydrogen atom viewer, http://falstad.com/qmatom/
g. Atom in a box simulation tool, http://daugerresearch.com/orbitals/index.shtml#download

## 1. List of Constants

| Property | Symbol | Value | Units |
|---|---|---|---|
| Mass of Electron | $m_e$ | $9.109 \times 10^{-31}$ | kg |
| Reduced Plank's Constant | $\hbar$ | $1.055 \times 10^{-34}$ | J·s |
| Elementary Charge | $e$ | $1.602 \times 10^{-19}$ | C |
| Permittivity of Free Space | $\varepsilon_0$ | $8.854 \times 10^{-12}$ | $C^2$/J.m |
| Bohr Radius for Hydrogen atom | $a$ | $5.292 \times 10^{-11}$ | m |

## 2. Chemistry Theory Segments

**Answer the following questions after Week 2 Cohort Lesson 1.**

**a.** Given the transformation from Cartesian ($x$, $y$, $z$) to spherical ($r$, θ, ϕ) coordinates are as follows:

$$x = r \sin\phi \cos\theta$$
$$y = r \sin\phi \sin\theta$$
$$z = r \cos\phi$$

(i) Express ($r$, θ, ϕ) in terms of ($x$, $y$, $z$).
(ii) Convert the spherical coordinates (3, 25°, 60°) to its Cartesian equivalent.

Answer and Workings:

**Answer the following questions after Week 3 Cohort Lesson 1.**

**b.** Given the formula for energy levels below, calculate the energy level in joules and eV for your chosen principal quantum number *n* = 2, 3 or 4.

$$\text{Energy level for nth quantum number}, E_n = -\left[\frac{m_e}{2\hbar^2}\left(\frac{e^2}{4\pi\mathcal{E}_0}\right)^2\right]\frac{1}{n^2}$$

Answer and Workings:

**c.** The Schrödinger equation is:

$$-\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi = E\Psi$$

For a system in spherical coordinates (*r*, θ, ϕ), the Laplacian operator in spherical coordinates is:

$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\left[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right]$$

The Laplaican operator is used to transform the above Schrödinger equation from Cartesian to spherical coordinates to give:

$$\frac{\partial}{\partial r}\left(r^2\frac{\partial\Psi}{\partial r}\right) + \frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Psi}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2\Psi}{\partial\phi^2} = (E - V)\left(-\frac{2mr^2}{\hbar^2}\right)\Psi$$

**d.** From the above equation in (c), notice that there are terms that depend only on *r* and terms that depend only on angles θ and ϕ. To solve the equation, we will need to apply separation of variables to derive the expressions for both the angular component (equation written in terms of θ and ϕ only) and radial component (equation written in terms of *r* only). This can be done through the use of a separation constant, $l(l+1)$, so that the LHS of the equation adds up to the RHS of the equation in (c). Do not substitute the chosen quantum numbers into the equation.

Hint:
We are looking at solutions that are separable into products in the form of:

$$\Psi(r,\theta,\phi) = R(r)\cdot Y(\theta,\phi)$$

You will need to use:

$$l(l+1) = \frac{1}{R}\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) + \frac{2mr^2}{\hbar^2}\left(E + \frac{e^2}{4\pi\varepsilon_0 r}\right) \text{ and } -V = \frac{e^2}{4\pi\varepsilon_0 r}$$

Answer and Workings:

$$\frac{\partial}{\partial r}\left(r^2\frac{\partial\Psi}{\partial r}\right)+\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Psi}{\partial\theta}\right)+\frac{1}{\sin^2\theta}\frac{\partial^2\Psi}{\partial\phi^2}=(E-V)\left(-\frac{2mr^2}{\hbar^2}\right)\Psi$$

The equation representing the radial component is:

The equation representing the angular component is:

**Answer the following questions together with Digital World after Week 3.**
Now that we have separated the equation in (c) to two equations in (d), we can solve for each component individually. With the chosen quantum numbers, solve for the angular component first in (e) and then the radial component in (f).

**e.** Given the normalized angular solution for Y(θ,ϕ), associated Legendre function and the Legendre polynomial, find the specific normalized solution for your chosen quantum numbers.

$$\text{Legendre Polynomial}, P_l(x)=\frac{1}{2^l l!}\left(\frac{\partial}{\partial x}\right)^l(x^2-1)^l$$

$$\text{Associated Legendre Function}, P_l^m(x)=(1-x^2)^{\frac{|m|}{2}}\left(\frac{\partial}{\partial x}\right)^{|m|}P_l(x)$$

$$\text{Normalized Angular Solution}, Y_l^m(\theta,\phi)=\epsilon\sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}e^{im\phi}P_l^m(\cos\theta)$$

$$\text{where } \epsilon=\begin{cases}(-1)^m \text{ for } m>0\\ 1 \text{ for } m\leq 0\end{cases}$$

Note: $P_l^m(\cos\theta)$ is a function of $\cos\theta$ while $P_l^m(x)$ is a function of $x$.

Answer and Workings:

**f.** Given the normalized radial solution for R($r$), associated Laguerre function and the Laguerre polynomial, find the specific normalized solution, in terms of $r$ and $a$, for your chosen quantum numbers. Do not substitute the value of $a$ into the solution.

$$\text{Laguerre Polynomial}, L_q(x) = e^x \left(\frac{\partial}{\partial x}\right)^q (e^{-x}x^q)$$

$$\text{Associated Laguerre Function}, L_{q-p}^p(x) = (-1)^p \left(\frac{\partial}{\partial x}\right)^p L_q(x)$$

$$\text{Normalized Radial Solution}, R_l^n(r) = \sqrt{\left(\frac{2}{na}\right)^3 \frac{(n-l-1)!}{2n[(n+l)!]^3}}\, e^{-\frac{r}{na}} \left(\frac{2r}{na}\right)^l \left[L_{q-p}^p\left(\frac{2r}{na}\right)\right]$$

where $p = 2l + 1$ and $q - p = n - l - 1$

Note: $L_{q-p}^p\left(\frac{2r}{na}\right)$ is a function of $\frac{2r}{na}$ while $L_{q-p}^p(x)$ is a function of $x$.

Answer and Workings:

**10.009 The Digital World**

Term 3. 2017

Problem Set 7 (for Chemistry Project)

Last update: January 12, 2017

Due dates:

- **Problems**: Monday, Week 9, 5:00pm.

**Objectives:**

1. Learn to use Numpy for numerical computations

**Note**: Solve the programming problems listed below using your favourite editor and test it. Make sure you save your programs in files with suitably chosen names and in an newly created directory. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

**Problems**

1. **Week 2:** Create a function to calculate the energy level of a given principal quantum number. This function should take 1 integer argument and return the energy level in eV. Round to 5 decimal places. (Pre-requisite: Part 2a) Hint: You can use import `scipy.constants` as c to get the necessary constants.

   To test:

   ```
   print 'n = 1'
   ans= energy_n(1)
   print ans

   print 'n = 2'
   ans= energy_n(2)
   print ans

   print 'n = 3'
   ans= energy_n(3)
   print ans
   ```

   The output should be:

   ```
   n = 1
   -13.60569
   n = 2
   -3.40142
   n = 3
   -1.51174
   ```

2. **Week 2:** Create two functions to convert degrees to radian and radian to degrees respectively. These functions should take 1 float argument and return the respective conversions each. Round to 5 decimal places.

   To Test:

   ```
   print 'deg_to_rad(90)'
   ans=deg_to_rad(90)
   print ans

   print 'deg_to_rad(180)'
   ans=deg_to_rad(180)
   print ans

   print 'deg_to_rad(270)'
   ans=deg_to_rad(270)
   print ans

   print 'rad_to_deg(3.14)'
   ans=rad_to_deg(3.14)
   print ans

   print 'rad_to_deg(3.14/2.0)'
   ans=rad_to_deg(3.14/2.0)
   print ans

   print 'rad_to_deg(3.14*3/4)'
   ```

```
ans=rad_to_deg(3.14*3/4)
print ans
```

The output should be:

```
deg_to_rad(90)
1.5708
deg_to_rad(180)
3.14159
deg_to_rad(270)
4.71239
rad_to_deg(3.14)
179.90875
rad_to_deg(3.14/2.0)
89.95437
rad_to_deg(3.14*3/4)
134.93156
```

3. **Week 2:** Create two functions to convert spherical to cartesian coordinates and cartesian to spherical coordinates. These functions should take 3 float arguments and return the 3 respective conversions. Round to 5 decimal places. (Pre-requisite: Part 2b) The convention is shown below.

   Hint: you can use Numpy trigonometric function by doing `import numpy as np`.

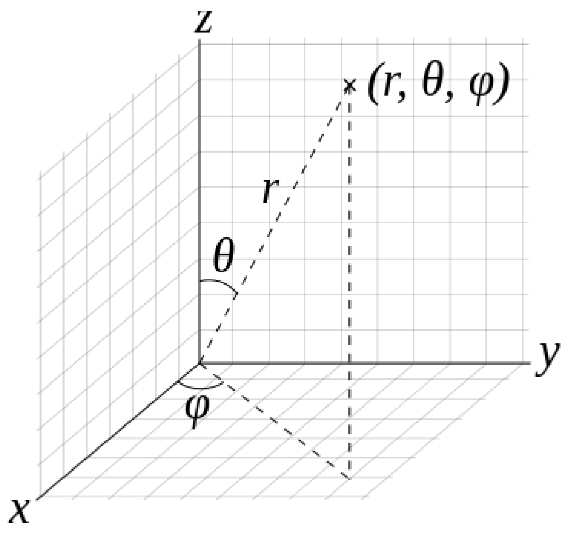

Figure 1: Spherical Coordinate System.

To test:

```
print 'spherical_to_cartesian(3,0,np.pi)'
ans=spherical_to_cartesian(3,0,np.pi)
print ans

print 'spherical_to_cartesian(3,np.pi/2.0,np.pi/2.0)'
ans=spherical_to_cartesian(3,np.pi/2.0,np.pi/2.0)
print ans
print 'spherical_to_cartesian(3,np.pi, 0)'
ans=spherical_to_cartesian(3,np.pi,0)
print ans

print 'cartesian_to_spherical(3,0,0)'
ans=cartesian_to_spherical(3,0,0)
print ans

print 'cartesian_to_spherical(0,3,0)'
ans=cartesian_to_spherical(0,3,0)
print ans

print 'cartesian_to_spherical(0,0,3)'
ans=cartesian_to_spherical(0,0,3)
print ans

print 'cartesian_to_spherical(0,-3,0)'
ans=cartesian_to_spherical(0,-3,0)
print ans
```

The output should be:

```
spherical_to_cartesian(3,0,np.pi)
(-0.0, 0.0, 3.0)
spherical_to_cartesian(3,np.pi/2.0,np.pi/2.0)
(0.0, 3.0, 0.0)
spherical_to_cartesian(3,np.pi, 0)
(0.0, 0.0, -3.0)
cartesian_to_spherical(3,0,0)
(3.0, 1.5708, 0.0)
cartesian_to_spherical(0,3,0)
(3.0, 1.5708, 1.5708)
cartesian_to_spherical(0,0,3)
(3.0, 0.0, 0.0)
cartesian_to_spherical(0,-3,0)
(3.0, 1.5708, -1.5708)
```

4. **Week 3:** Create a function using the while loop that calculates the factorial of the input integer. For example, 4! = 4 x 3 x 2 x1 = 24. This function should take 1 integer argument and return the calculated factorial. The return value should be an integer.

   To test:

```
print 'fact(3)'
ans=fact(3)
print ans


print 'fact(5)'
ans=fact(5)
print ans


print 'fact(4)'
ans=fact(4)
print ans


print 'fact(1)'
```

```
ans=fact(1)
print ans
```

The output should be:

```
fact(3)
6
fact(5)
120
fact(4)
24
fact(1)
1
```

5. **Week 3:** Create a function that calculates the associated Legendre function. This function should take in 2 integer arguments and return the function that computes the associated Legendre with a variable $\cos(\theta)$. You should be able to obtain the function for any combination of m=0,1,2,3 and n=0,1,2,3 (Pre-requisite: Part 2e).

   Hint: You may want to create a function that acts as a reference table to a set of recalculated associated Legendre function. For example,

```
def p00(theta):
  return 1

def assoc_legendre(m,l):
  if m==0 and l==0:
    return p00
```

   To test:

```
print 'f=assoc_legendre(0,0)'
print 'f(1)'
f=assoc_legendre(0,0)
ans=f(1)
print ans

print 'f=assoc_legendre(1,1)'
print 'f(1)'
f=assoc_legendre(1,1)
ans=f(1)
print ans

print 'f=assoc_legendre(2,3)'
print 'f(1)'
f=assoc_legendre(2,3)
ans=f(1)
print ans

print 'f=assoc_legendre(2,3)'
print 'f(0)'
f=assoc_legendre(2,3)
ans=f(0)
print ans
```

   The output should be:

```
f=assoc_legendre(0,0)
f(1)
1
f=assoc_legendre(1,1)
f(1)
0.841470984808
f=assoc_legendre(2,3)
f(1)
5.73860550926
f=assoc_legendre(2,3)
f(0)
0.0
```

6. **Week 3:** Create a function that calculates the associated Laguerre function. This function should take in 2 integer arguments and return the function that computes the associated Laguerre with a variable x. You should be able to obtain the function for any combination of $p = 0, 1, 2, 3$ and $q - p = 0, 1, 2, 3$ (Pre-requisite: Part 2f)

   Hint: You may want to create a function that acts as a reference table to a set of recalculated associated Laguerre function. For example

   ```
   def l00(x):
     return 1

   def assoc_laguerre(p,qmp):
     if p==0 and qmp==0:
       return l00
   ```

   Note that `qmp` is the argument for $q - p$.

   To test:

   ```
   print 'f=assoc_laguerre(0,0)'
   print 'f(1)'
   f=assoc_laguerre(0,0)
   ans=f(1)
   print ans

   print 'f=assoc_laguerre(1,1)'
   print 'f(1)'
   f=assoc_laguerre(1,1)
   ans=f(1)
   print ans

   print 'f=assoc_laguerre(2,2)'
   print 'f(1)'
   f=assoc_laguerre(2,2)
   ans=f(1)
   print ans

   print 'f=assoc_laguerre(2,2)'
   print 'f(0)'
   f=assoc_laguerre(2,2)
   ans=f(0)
   print ans
   ```

   The output should be:

```
f=assoc_laguerre(0,0)
f(1)
1
f=assoc_laguerre(1,1)
f(1)
2
f=assoc_laguerre(2,2)
f(1)
60
f=assoc_laguerre(2,2)
f(0)
144
```

7. **Week 4:** Create a function that calculates the normalized angular solution. This function should take 4 float arguments and return the value of the normalized angular solution for the specific m, l, $\theta$ and $\phi$. (Pre-requisite: Part 2e). The return value is a complex number rounded to 5 decimal places for both the real and the imaginary parts. Hint: You may want to use `np.round()` function to round the return value to 5 decimal places. You can use the previous function `assoc_legendre(m,l)`, in the following way:

   ```
   pfunc=assoc_legendre(m,l)
   y=pfunc(theta)
   ```

   where $m$ and $l$ are the respective quantum numbers and theta is the angle $\theta$. This means that `assoc_legendre(m,l)` must return a function with one argument $\theta$. See the test cases on the `assoc_legendre(m,l)` question.

   To test:

   ```
   print 'angular_wave_func(0,0,0,0)'
   ans=angular_wave_func(0,0,0,0)
   print ans

   print 'angular_wave_func(0,1,c.pi,0)'
   ans=angular_wave_func(0,1,c.pi,0)
   print ans

   print 'angular_wave_func(1,1,c.pi/2,c.pi)'
   ans=angular_wave_func(1,1,c.pi/2,c.pi)
   print ans

   print 'angular_wave_func(0,2,c.pi,0)'
   ans=angular_wave_func(0,2,c.pi,0)
   print ans
   ```

   The output should be:

   ```
   angular_wave_func(0,0,0,0)
   (0.28209+0j)
   angular_wave_func(0,1,c.pi,0)
   (-0.4886+0j)
   angular_wave_func(1,1,c.pi/2,c.pi)
   (0.34549+0j)
   angular_wave_func(0,2,c.pi,0)
   (0.63078+0j)
   ```

8. **Week 4:** Create a function that calculates the normalized radial solution. This function should take 3 float arguments and return the value of the normalized radial solution. The return value should be normalized to $a^{-3/2}$, where $a$ is the Bohr's radius, and rounded to 5 decimal places. (Pre-requisite: Part 2f). Hint: You may want to use `np.round()` function to round the return value to 5 decimal places. You can use the previous function `assoc_laguerre(p, qmp)`, in the following way:

   ```
   lfunc=assoc_laguerre(p, qmp)
   y=lfunc(x)
   ```

   where the argument $p$ and $qmp$ refers to $p$ and $q-p$ in the associated Laguerre. This means that `assoc_laguerre(n,l,r)` must return a function with one argument. See the test cases on the `assoc_laguerre(p, qmp)` question.

   To test:

   ```
   a=c.physical_constants['Bohr radius'][0]
   print 'radial_wave_func(1,0,a)'
   ans=radial_wave_func(1,0,a)
   print ans

   print 'radial_wave_func(1,0,a)'
   ans=radial_wave_func(1,0,a)
   print ans

   print 'radial_wave_func(2,1,a)'
   ans=radial_wave_func(2,1,a)
   print ans

   print 'radial_wave_func(2,1,2*a)'
   ans=radial_wave_func(2,1,2*a)
   print ans
   ```

   The output should be:

   ```
   radial_wave_func(1,0,a)
   0.73576
   radial_wave_func(1,0,a)
   0.73576
   radial_wave_func(2,1,a)
   0.12381
   radial_wave_func(2,1,2*a)
   0.15019
   radial_wave_func(3,1,2*a)
   0.08281
   ```

9. **Week 5:** Create a function that calculates the square of the magnitude of the real wave function. The function takes in several arguments:

   - $n$: quantum number $n$
   - $l$: quantum number $l$
   - $m$: quantum number $m$

- *roa*: maximum distance to plot from the centre, normalized to Bohr radius, i.e. $r/a$.
- $N_x$: Number of points in the positive $x$ axis.
- $N_y$: Number of points in the positive $y$ axis.
- $N_z$: Number of points in the positive $z$ axis.

The function should returns:

- *xx*: $x$ location of all the points in a 3D Numpy array.
- *yy*: $y$ location of all the points in a 3D Numpy array.
- *zz*: $z$ location of all the points in a 3D Numpy array.
- *density*: The square of the magnitude of the real wave function, i.e. $|\Psi|^2$

Note that when $m! = 0$, the real wave function is a linear combination of two stationary states. For example, for $n = 2, l = 1, m = 1$,

$$\Psi(r,0) = \frac{1}{\sqrt{2}}(\Psi_{211} + \Psi_{21-1})$$

Hint: You may find the following functions to be useful:

- `fvec=numpy.vectorize(f)`: This function takes in a function and return its vectorized version of the function.
- `xx,yy,zz=numpy.meshgrid(x,y,z)`: This function takes in 1D arrays and returns its 3D arrays to conform to a 3D grid.
- `m=numpy.absolute(c)`: This function takes in a complex number and returns its absolute value or its magnitude.

Hint: You also need to use all the previous functions you have done. Note that some of those functions may round the output to 5 decimal places and the final magnitude output from this function should also be rounded to 5 decimal places.

To test:

```
print 'Test 1'
x,y,z,mag=hydrogen_wave_func(2,1,1,8,2,2,2)
print 'x, y, z:'
print x, y, z
print 'mag:'
print mag

print 'Test 2'
x,y,z,mag=hydrogen_wave_func(2,1,1,5,3,4,2)
print 'x, y, z:'
print x, y, z
print 'mag:'
print mag
print 'Test 3'
x,y,z,mag=hydrogen_wave_func(2,0,0,3,5,4,3)
print 'x, y, z:'
print x, y, z
print 'mag:'
print mag
```

The output should be:

```
Test 1
x, y, z:
[[[-8. -8.]
  [ 8.  8.]]

 [[-8. -8.]
  [ 8.  8.]]] [[[-8. -8.]
  [-8. -8.]]

 [[ 8.  8.]
  [ 8.  8.]]] [[[-8.  8.]
  [-8.  8.]]

 [[-8.  8.]
  [-8.  8.]]]
mag:
[[[ 0.  0.]
  [ 0.  0.]]

 [[ 0.  0.]
  [ 0.  0.]]]
Test 2
x, y, z:
[[[-5. -5.]
  [ 0.  0.]
  [ 5.  5.]]

 [[-5. -5.]
  [ 0.  0.]
  [ 5.  5.]]

 [[-5. -5.]
  [ 0.  0.]
  [ 5.  5.]]

 [[-5. -5.]
  [ 0.  0.]
  [ 5.  5.]]] [[[-5.       -5.      ]
  [-5.       -5.      ]
  [-5.       -5.      ]]

 [[-1.66667 -1.66667]
  [-1.66667 -1.66667]
  [-1.66667 -1.66667]]

 [[ 1.66667  1.66667]
  [ 1.66667  1.66667]
  [ 1.66667  1.66667]]

 [[ 5.        5.      ]
```

```
  [ 5.        5.      ]
  [ 5.        5.      ]]] [[[-5.  5.]
  [-5.  5.]
  [-5.  5.]]

 [[-5.  5.]
  [-5.  5.]
  [-5.  5.]]

 [[-5.  5.]
  [-5.  5.]
  [-5.  5.]]

 [[-5.  5.]
  [-5.  5.]
  [-5.  5.]]]
mag:
[[[  9.00000000e-05   9.00000000e-05]
  [  0.00000000e+00   0.00000000e+00]
  [  9.00000000e-05   9.00000000e-05]]

 [[  3.50000000e-04   3.50000000e-04]
  [  0.00000000e+00   0.00000000e+00]
  [  3.50000000e-04   3.50000000e-04]]

 [[  3.50000000e-04   3.50000000e-04]
  [  0.00000000e+00   0.00000000e+00]
  [  3.50000000e-04   3.50000000e-04]]

 [[  9.00000000e-05   9.00000000e-05]
  [  0.00000000e+00   0.00000000e+00]
  [  9.00000000e-05   9.00000000e-05]]]
Test 3
x, y, z:
[[[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]

 [[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]

 [[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]

 [[-3.  -3.  -3. ]
  [-1.5 -1.5 -1.5]
  [ 0.   0.   0. ]
  [ 1.5  1.5  1.5]
  [ 3.   3.   3. ]]] [[[-3. -3. -3.]
  [-3. -3. -3.]
  [-3. -3. -3.]
  [-3. -3. -3.]
  [-3. -3. -3.]]
 [[-1. -1. -1.]
  [-1. -1. -1.]
  [-1. -1. -1.]
  [-1. -1. -1.]
  [-1. -1. -1.]]

 [[ 1.  1.  1.]
  [ 1.  1.  1.]
  [ 1.  1.  1.]
  [ 1.  1.  1.]
  [ 1.  1.  1.]]

 [[ 3.  3.  3.]
  [ 3.  3.  3.]
  [ 3.  3.  3.]
  [ 3.  3.  3.]
  [ 3.  3.  3.]]] [[[-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]]

 [[-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]]

 [[-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]]

 [[-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]
  [-3.  0.  3.]]]
mag:
[[[  5.60000000e-04   7.20000000e-04   5.60000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  7.20000000e-04   5.00000000e-04   7.20000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  5.60000000e-04   7.20000000e-04   5.60000000e-04]]

 [[  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  5.70000000e-04   3.66000000e-03   5.70000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]]

 [[  7.10000000e-04   5.70000000e-04   7.10000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  5.70000000e-04   3.66000000e-03   5.70000000e-04]
  [  6.80000000e-04   6.00000000e-05   6.80000000e-04]
  [  7.10000000e-04   5.70000000e-04   7.10000000e-04]]

 [[  5.60000000e-04   7.20000000e-04   5.60000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  7.20000000e-04   5.00000000e-04   7.20000000e-04]
  [  6.90000000e-04   6.40000000e-04   6.90000000e-04]
  [  5.60000000e-04   7.20000000e-04   5.60000000e-04]]]
```

10. **Week 8: Plots:** Submit your plot for your assigned quantum numbers to your Chemistry instructors to get a point for this item.

11. **Week 8:** In the final function to calculate the hydrogen wave function, you are to use the other previous functions you have calculated. However, some of those functions rounds the result to 5 decimal places. The error on the final wave function magnitude is called ____ due to ____.

    (a) floating point error, rounding error.

    (b) propagation error, rounding error.

    (c) propagation error, floating point error.

    (d) rounding error, propagation error.

    **Submit your answer on Tutor.**

12. **Week 8:** What is the effect when you increase the number of points $Nx, Ny, Nz$, while maintaining the values the other parameters?

    (a) increase of accuracy, decrease of computational time.

    (b) decrease of accuracy, increase of computational time.

    (c) increalse of accuracy, increase of computational time.

    (d) decrease of accuracy, decrease of computational time.

    **Submit your answer on Tutor.**

13. **Week 8:** What is the effect of increasing the distance $r/a$, while maintaining the values of the other paramters?

    (a) increase of accuracy, no change in computational time.

    (b) decrease of accuracy, change in computational time.

    (c) increalse of accuracy, change in computational time.

    (d) decrease of accuracy, no change in computational time.

    **Submit your answer on Tutor.**

**Plotting sample codes**:

- You can use the following code to save the Python data to a file:

```
import numpy as np

#######
# write all your function definitions here
#######

x,y,z,mag=hydrogen_wave_func(3,1, 0,10,20,20,20)

x.dump('xdata310.dat')
y.dump('ydata310.dat')
z.dump('zdata310.dat')
mag.dump('density310.dat')
```

- You can use the following code to plot using matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.load('xdata310.dat')
y = np.load('ydata310.dat')
z = np.load('zdata310.dat')

mag = np.load('density310.dat')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for a in range(0,len(mag)):
    for b in range(0,len(mag)):
        for c in range(0,len(mag)):
            ax.scatter(x[a][b][c],y[a][b][c],z[a][b][c], marker='o',
                alpha=(mag[a][b][c]/np.amax(mag)))
plt.show()
```

- You can use the following code to plot using mlab Mayavi package:

```
import numpy as np
from mayavi import mlab

x = np.load('xdata310.dat')
y = np.load('ydata310.dat')
z = np.load('zdata310.dat')

density = np.load('density310.dat')

figure = mlab.figure('DensityPlot')
mag=density/np.amax(density)

pts = mlab.points3d(mag,opacity=0.5, transparent=True)
# or pts = mlab.contour3d(mag, opacity=0.5)

mlab.colorbar(orientation='vertical')
mlab.axes()
mlab.show()
```

**Plots**

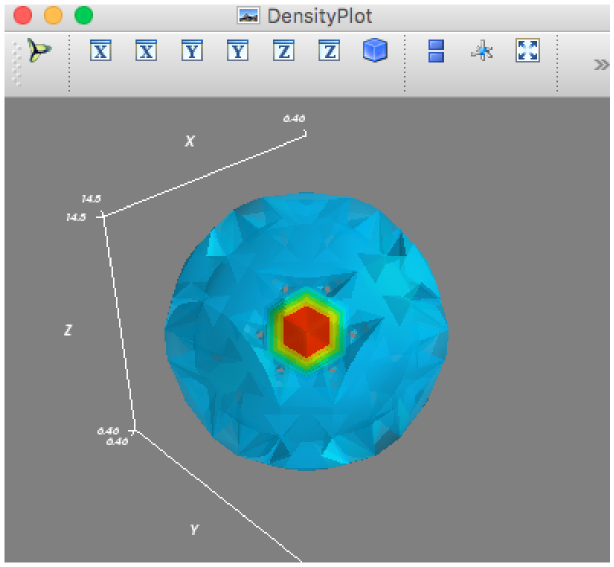

Figure 2: Magnitude plot for $n = 2, l = 0, m = 0$ using contour3d from mlab Mayavi package.

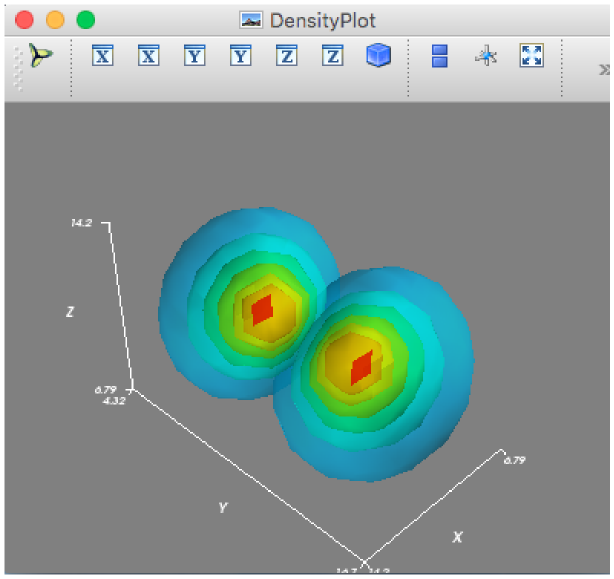

Figure 3: Magnitude plot for $n = 2, l = 1, m = 1$ using contour3d from mlab Mayavi package.

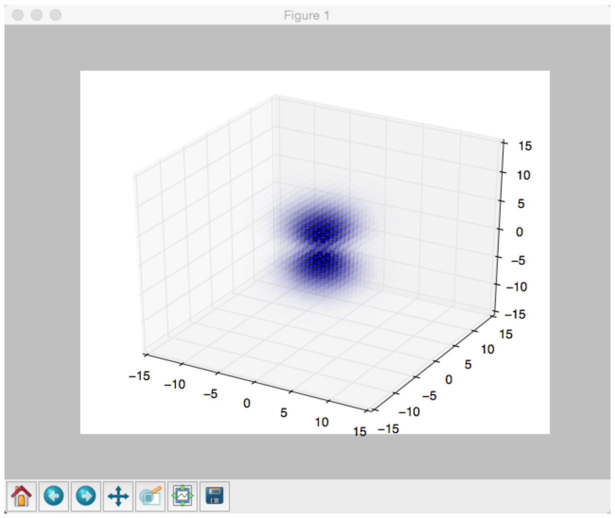

Figure 4: Magnitude plot for $n = 2, l = 1, m = 0$ using scatter from Matplotlib package.

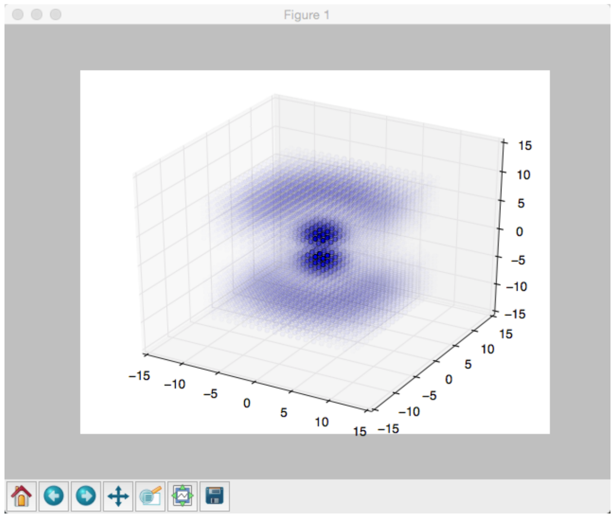

Figure 5: Magnitude plot for $n = 3, l = 1, m = 0$ using scatter from Matplotlib package.

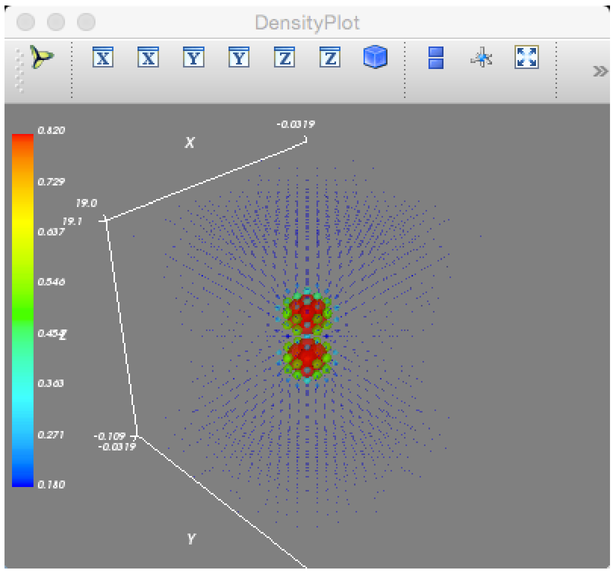

Figure 6: Magnitude plot for $n = 3, l = 1, m = 0$ using points3d from mlab Mayavi package.

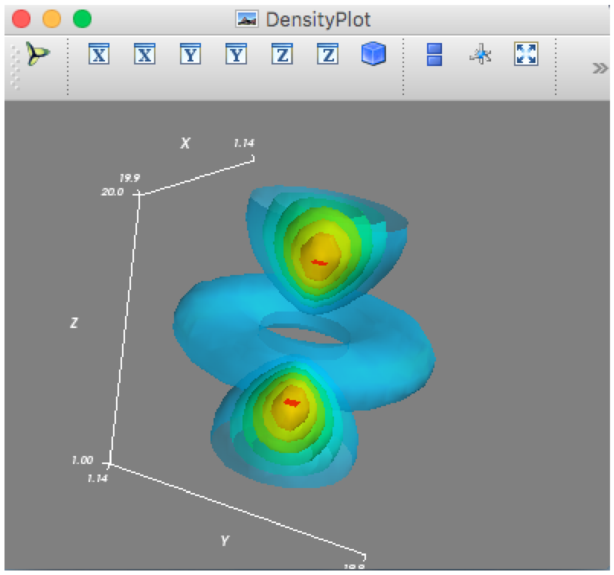

Figure 7: Magnitude plot for $n = 3, l = 2, m = 0$ using contour3d from mlab Mayavi package.

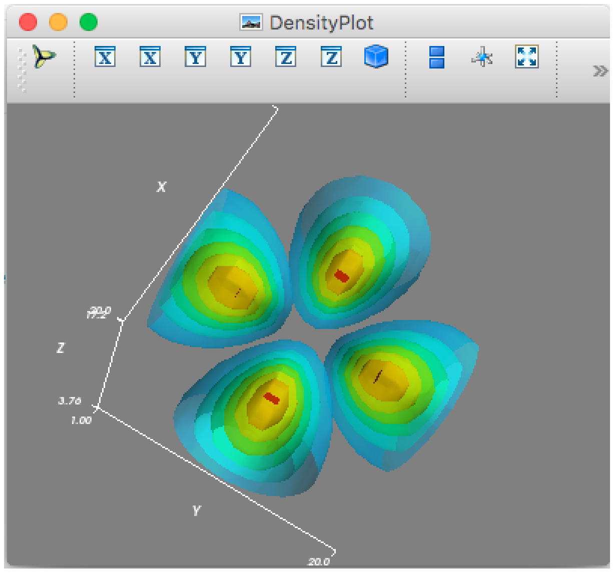

Figure 8: Magnitude plot for $n = 3, l = 2, m = 2$ using contour3d from mlab Mayavi package.

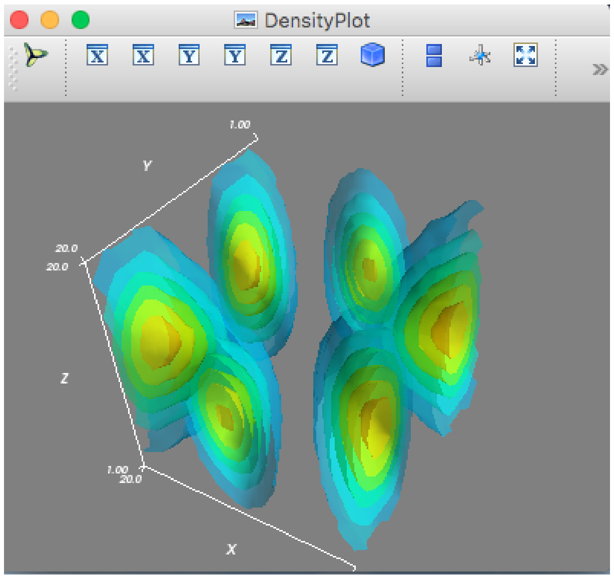

Figure 9: Magnitude plot for $n = 4, l = 3, m = 3$ using contour3d from mlab Mayavi package.